

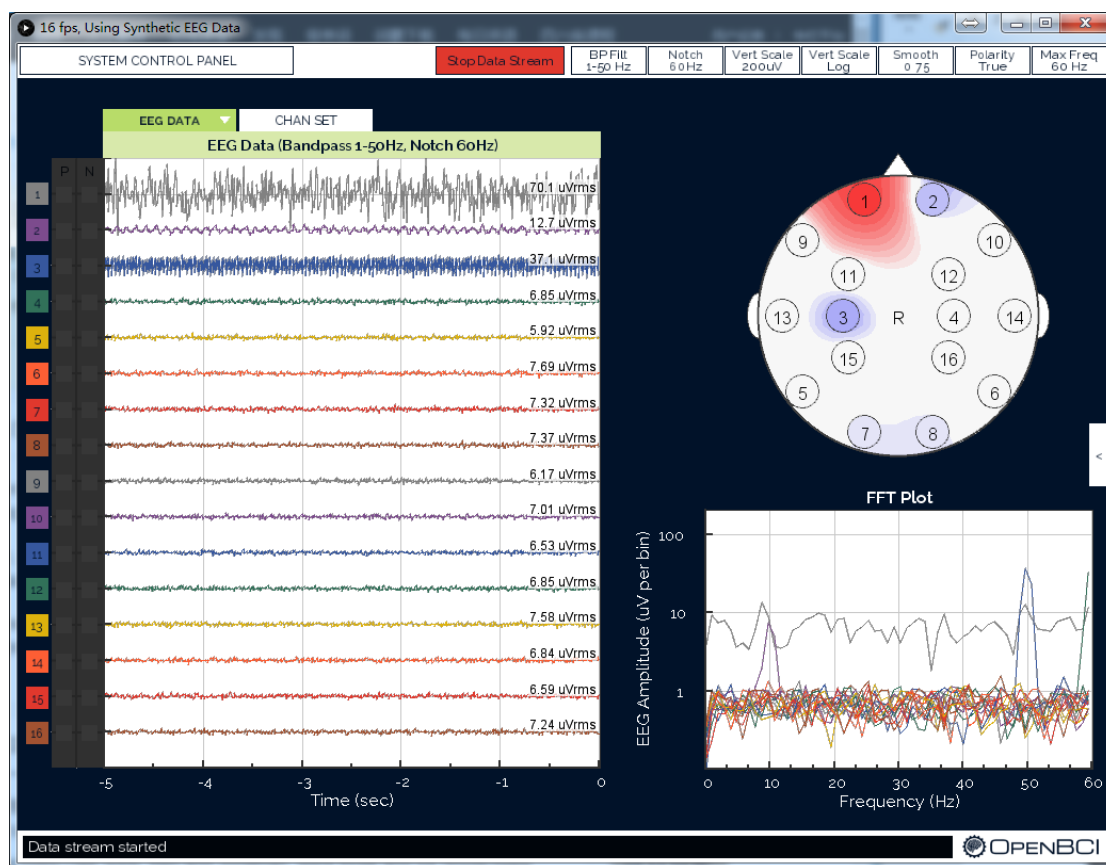
## Brainwave Sensor for the TDR System

### Introduction

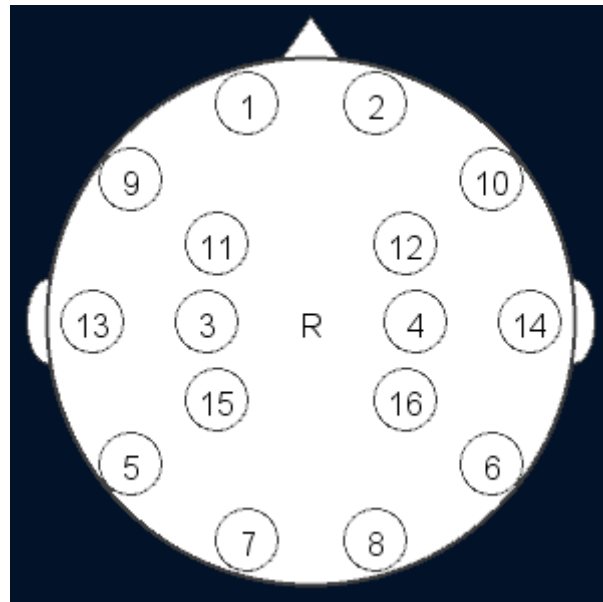
In this project, I worked on adding new sensor in to the TDR system. What I have done are connect OpenBCI brainwave sensor into the system. In addition, I also developed a general program to predict the status of brainwave in real time.

### OpenBCI

OpenBCI is an open source brain computer interface platform. OpenBCI boards support EEG, EMG, and EKG recording and measurement. It supports standard EEG electrodes, so it is easy to buy accessory for the board and let it start working. In addition, the OpenBCI community provides an open source OpenBCI GUI to work with OpenBCI boards, so I don't have to pay a lot of attention on how to record data from the board, but I can focus on how to send data to TDR system, and let the system monitor the incoming data. It provides us a change to focus on higher level data manipulation and computation. In addition, OpenBCI GUI has a Java implementation, so I can easily to integrate the GUI into the TDR system as a sensor component. The following picture shows a working OpenBCI GUI screenshot.



The version of our OpenBCI boards support 16 channels data recording. That is, the system can monitor EEG signal from different 16 position on the head at the same time. It provides us more data for further analysis, especially in machine learning process, more data always represent a better prediction model. The following picture shows the 16 channels and corresponding position on the head, respectively.

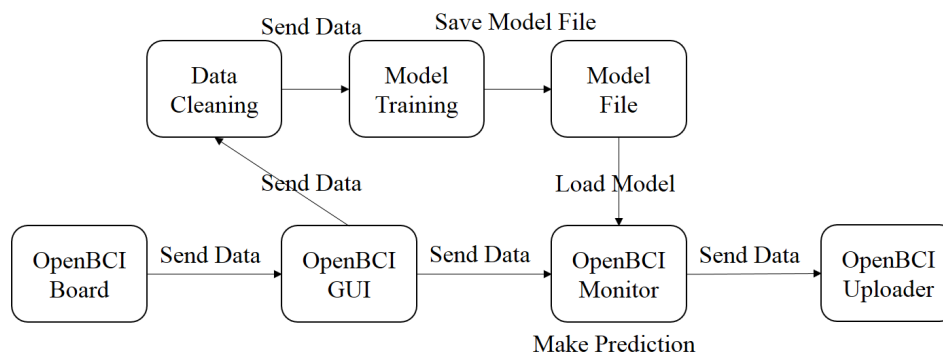


## Software Design

In this project, the OpenBCI board will connect to OpenBCI GUI. The OpenBCI GUI is a sensor component, and it sends data to OpenBCI Monitor component. The OpenBCI Monitor component makes prediction based on pre-trained prediction model, and decides if it is necessary to upload data into the database. If so, it sends a message to OpenBCI Uploader component. Once the OpenBCI Uploader component receives the message, it uploads the record into the database.

Out of a running TDR system, there is another program for cleaning plain data, training prediction model, and save the model as a file so that the OpenBCI Monitor can load this model and make prediction easily.

The following picture shows the working flow of this project.



## TDR System Components

In this section, I will talk about all components in this project that related to the TDR system, and this is the original task of this project. I followed the standard of the TDR system to implement each component, so that they can be integrate into the TDR system easily.

### OpenBCI Board

This is the physical sensor provided by the OpenBCI community. It supports 16 channels EEG signals monitor. We assembled a headset, so that user can wear it can don't need to worry about corresponding monitor position by themselves. Furthermore, to make it easy to use, an additional Bluetooth transmission board used to support wireless communication between OpenBCI board and OpenBCI GUI, so that user not limited by the cable connection between the OpenBCI headset and the computer.

### OpenBCI GUI

This GUI is also provided by the OpenBCI community. It visualizes the data from OpenBCI board. From the screenshot from the lase section, we can see that, this interface shows data from 16 channels separately, a frequency plot, and a head plot. User can interpret their status even only from these graphs. In addition, user can also manipulate coming data directly using this interface. Also, it saves data into a file directly, so that the machine learning module can use these data for prediction model training in the future.

This interface was written in Processing language, it is an alternative Java language, so I can modify it to support TDR system. In this project, this interface is a sensor component of the TDR system. It not only saves data into a file, but also sends a message contains all data from each channel to the TDR system. Thus, the OpenBCI Monitor can capture this message and make prediction or decide whether to let uploader to upload this record or not.

### OpenBCI Monitor

**##### Extra Deeds in Details Start #####**

Once the user start the OpenBCI Monitor, it loads the pre-trained model file for making prediction of coming data. The benefit of this is that, the system doesn't have train a new model every time, and it makes the whole process asynchronies. It not necessary to train a new model every time if we don't have high quality annotated data. Thus, before the machine learning module gets new annotated data, the OpenBCI Monitor just simply uses a pre-trained model for make prediction of meditation status in real time. The result is in percentage format, and it is a fuzzy representation other than just simple 1 or 0. I think this is better because sometime it is hard to simply to describe the meditation status of a user even manually.

**##### Extra Deeds in Details End #####**

Because the sample rate of the interface is 250 Hz, and it may be too much for the TDR system database, so the OpenBCI Monitor only lets the OpenBCI Uploader to upload records every second. User can adjust this interval easily by modifying the parameter.

### OpenBCI Uploader

The OpenBCI Uploader is a standard uploader in the TDR system. Like other uploader in the system, it receives message from other components, and inserts one or more records into the database on demand.

## ##### Extra Deeds in Details Start #####

### Machine Learning Module

In this section, I will talk about the machine learning module I implemented for this project. I used an external package named “Weka” for training prediction model. Weka is a powerful tool that provides a lot of flexible well-programmed machine learning algorithms. With its help, the module can train a prediction model easily using different machine learning algorithms. In addition, Weka is a package implemented by Java, so there is a chance to integrate the package into the TDR system directly as a component, but not just an external module.

### Data Cleaning

Before feed plain data to the Weka classifier, it is necessary to clean data, and generates a valid data set for Weka classifier. In original plain data, there are few lines of meta data before the actual records, it is necessary to delete those meta data. In addition, there is an index number before each record, it also needs to be eliminated.

Since my job is training prediction model using statistical machine learning algorithm, thus I need to extract features for training by myself. It is too easy to just use data from 16 channels as a training data, because they are brainwave records, and move up and down. Thus, these simple features cannot split data with different meditation status. I decided to add more features in it, to achieve this target, I combined 4 consecutive records together. The benefit I can get is that, other than  $16 * 4$  data from 16 channels, I can calculate mean, variance, and standard deviation for each channel. Now, we have  $16 * 4 + 3 * 16$  features. Since the sample rate is 250 Hz, we still can generate 62 records for training in only 1 second. I think this is enough for our task so far, if not, it is easy to be adjusted by modified one parameter in the module.

After construct all records, the model saves all records into an “arff” file, so that the system can feed this file into the Weka classifier directly for prediction model training.

### Model Training

Weka provides a lot of machine learning algorithms, from KNN or linear regression, to other high level machine learning algorithms, such as SVM. With the help of the package, the system can train any model by different classifiers. I implemented the program that, user can easily to change the classifier they need.

To select a proper algorithm, I did few experiments using different classifiers, such as Linear Regression, Logistic Regression, Bagging, Ada Boost, Naïve Bayes, J48, Random Forest, and SMV. I used 10-folds cross validate to select a better model. Unfortunately, the data I had are too far away from each other. Every classifier from simple to complicated, the accuracy is 100%. I cannot compare which model is better, so I just use SMV as the model.

Another task I want to do is to select as few as possible of data to train a model, so that users do not need to get a device to monitor 16 channels. A smaller and cheaper device that only monitor

few channels may be enough for meditation status prediction task. To select few channels, I used information gain for feature selection. Since my records are mostly plain data from each channel, thus, by using feature selection technique, it can select channels that can split data efficiently. Another solution is Principal Component Analysis (PCA). With help of PCA, the system can find out which feature (channel) contributes more to the data. Then we may use one or more most contributes channels to be our final channels. I also did few experiments on this task, with the same reason I got in classification part, the algorithm told me I can only use one channel for prediction, because the value of that channel will be positive if the record is positive, and the value will be negative if the record is negative. It still has 100% accuracy, but I need more data to verify this result.

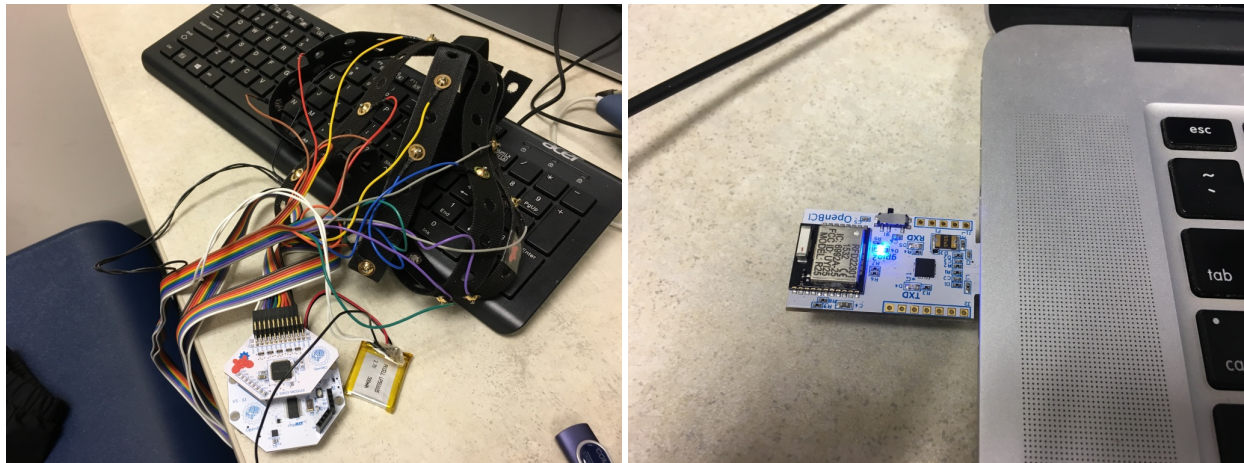
### Model File

After train a model, the module simple save the classifier model in to a file so that the OpenBCI Monitor can load this model and make prediction.

**##### Extra Deeds in Details End #####**

### Screenshots Demo

First, let a user wear the device, and plug in the Bluetooth communication device into the computer.



Then start SIS server, initialize it.

```
Scripts — java • sh runSISServer.sh — 80x24
Component Type: Basic
Name: SocialNetwork

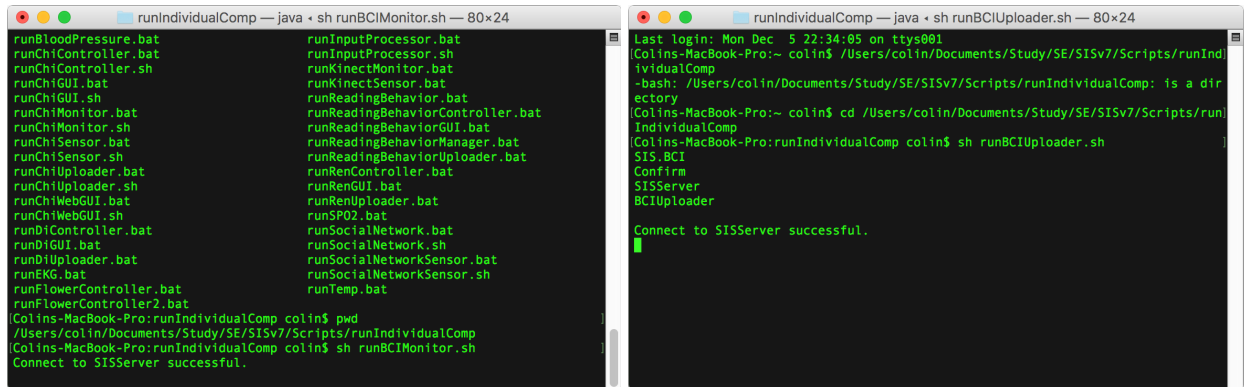
Scope: SIS.Chi
Component Type: Basic
Name: SocialNetworkSensor

Scope: SIS.Ren
Component Type: Basic
Name: SP02

Scope: SIS.D1
Component Type: Basic
Name: Temp

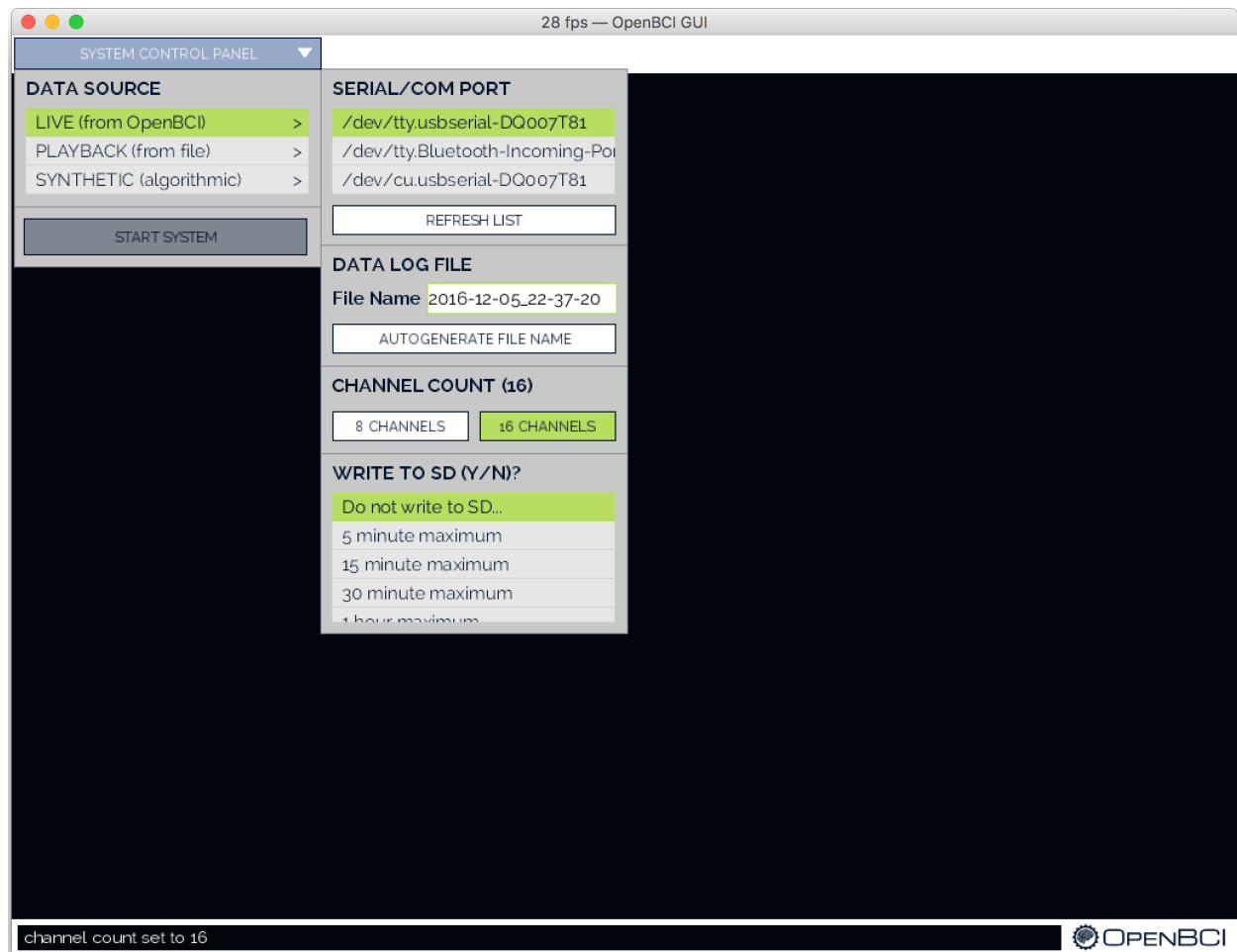
Scope: SIS.Chi
Component Type: Basic
Name: ChiMonitor
```

## Run BCI Monitor, and BCI Uploader

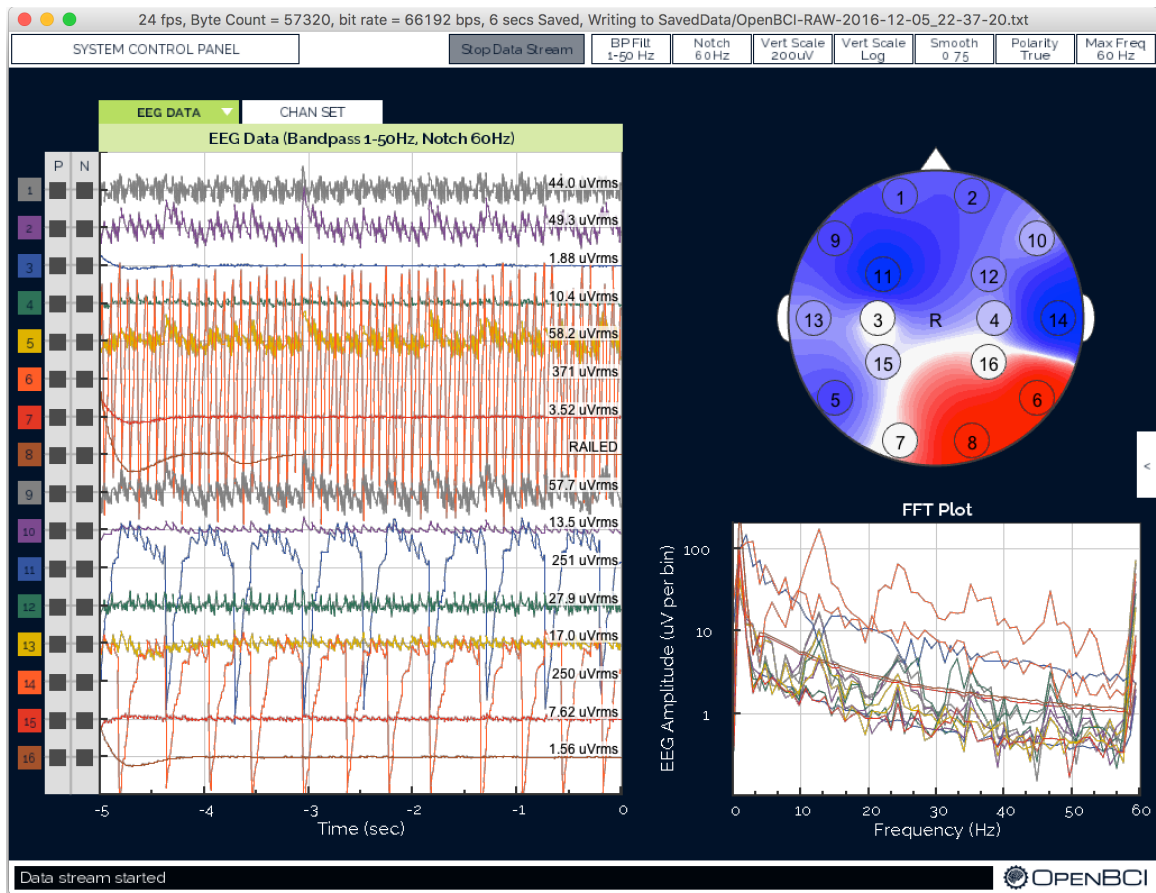


The image shows two terminal windows side-by-side. The left window, titled 'runIndividualComp — java • sh runBCIMonitor.sh — 80x24', displays a list of scripts to be run, including runBloodPressure.bat, runCh1Controller.bat, runCh1GUI.bat, runCh1Monitor.bat, runCh1Sensor.bat, runCh1Uploader.bat, runCh1WebGUI.bat, runDiController.bat, runDiGUI.bat, runDiUploader.bat, runEKG.bat, runFlowController.bat, runFlowController2.bat, runInputProcessor.bat, runInputProcessor.sh, runKinectMonitor.bat, runKinectSensor.bat, runReadingBehavior.bat, runReadingBehaviorController.bat, runReadingBehaviorGUI.bat, runReadingBehaviorManager.bat, runReadingBehaviorUploader.bat, runRenController.bat, runRenGUI.bat, runRenUploader.bat, runSP02.bat, runSocialNetwork.bat, runSocialNetworkSensor.bat, runSocialNetworkSensor.sh, and runTemp.bat. The right window, titled 'runIndividualComp — java • sh runBCIUploader.sh — 80x24', shows the output of the uploader script, including the command 'sh runBCIUploader.sh' and the message 'Connect to SISServer successful.'

Open OpenBCI GUI, and select “LIVE (from OpenBCI)”, and proper parameters. Then click “Start System”



Click “Start Data Stream” to read data from OpenBCI board.



From the BCI Uploader, we can see that, data were uploaded to the database and send an email to the database.

```
runIndividualComp — java < sh runBCIUploader.sh — 80x24

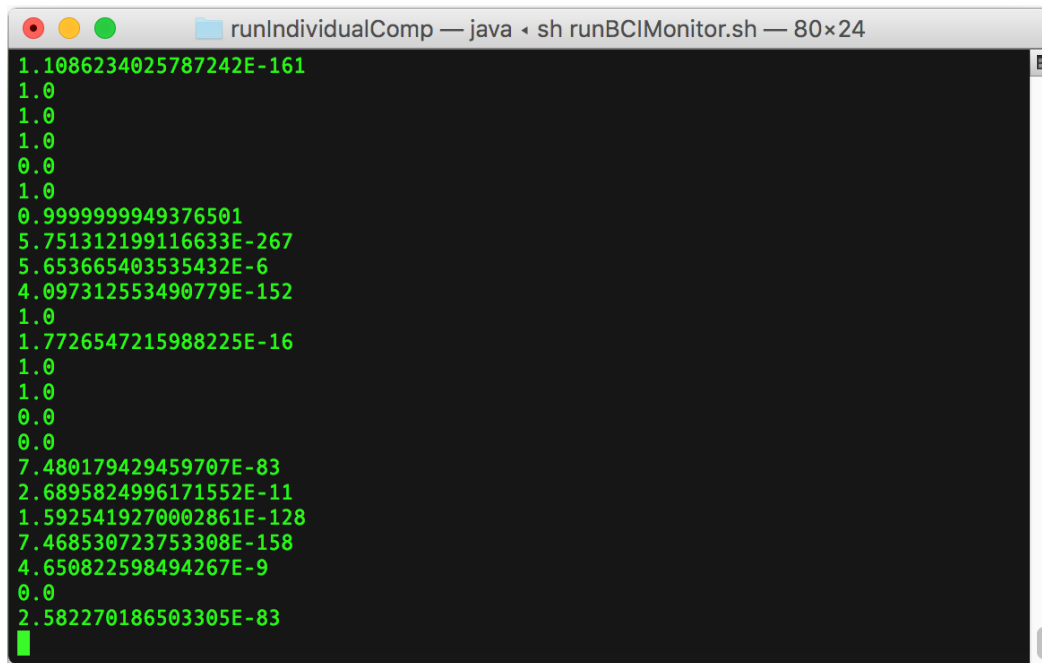
Channel 0: -29330.494140625
Channel 1: -28797.80859375
Channel 2: 40582.69921875
Channel 3: -28742.107421875
Channel 4: -30083.16796875
Channel 5: -29344.955078125
Channel 6: 90620.671875
Channel 7: 187500.0
Channel 8: -29410.513671875
Channel 9: -36218.4296875
Channel 10: -29587.986328125
Channel 11: -28809.095703125
Channel 12: -28625.4765625
Channel 13: -29472.62890625
Channel 14: -29033.037109375
Channel 15: 125992.6640625
UserID: 376896
Date (BCI): 1480995480960
-----

Updating DB...
DB Updated
```



### ##### Extra Deeds in Details Start #####

From the BCI Monitor, we can see that, the system give a probability of user are in a meditation status.



```
runIndividualComp — java < sh runBCIMonitor.sh — 80x24
1.1086234025787242E-161
1.0
1.0
1.0
0.0
1.0
0.9999999949376501
5.751312199116633E-267
5.653665403535432E-6
4.097312553490779E-152
1.0
1.7726547215988225E-16
1.0
1.0
0.0
0.0
7.480179429459707E-83
2.6895824996171552E-11
1.5925419270002861E-128
7.468530723753308E-158
4.650822598494267E-9
0.0
2.582270186503305E-83
█
```

Above are live demo screenshots. In this project, we also can train, test a model, and save the model into a file, so that the BCI Monitor can load the model easily and make prediction in the live demo.

If user want to try another model, just simply modify the code, to select any classifiers they want to try.

```
public void TrianTest(Instances trainset, Instances testset) throws Exception{
    Classifier[] classifiers = {
        // new Bagging(),
        // new AdaBoostM1(),
        //new IBk(),
        //new IBk(3),
        //new IBk(5),
        //new NaiveBayes(),
        //new J48(),
        new SMO(),
        //new LinearRegression(),
        //new Vote(),
        //new RandomForest(),
    };

    for(int i = 0; i < classifiers.length; i++)
    {
        Classifier classifier = classifiers[i];
        //RandomForest classifier = (RandomForest) classifiers[i];
        //classifier.setMaxDepth(5);
        classifier.buildClassifier(trainset);
        //System.out.println(classifier);
    }
}
```



After select the classifier, you can run the program to do a 10-folds cross validation on selected classifier, and output the result.

```

          TP Rate  FP Rate  Precision  Recall  F-Measure  ROC Area  Class
          1         0         1         1         1         1         0
          1         0         1         1         1         1         1
Weighted Avg.  1         0         1         1         1         1
=== Confusion Matrix ===
      a  b  <-- classified as
30  0  |  a = 0
 0 21  |  b = 1

train:/Users/colin/Documents/Study/SE/data_08
train: 461
test: 51
=== Detailed Accuracy By Class ===
          TP Rate  FP Rate  Precision  Recall  F-Measure  ROC Area  Class
          1         0         1         1         1         1         0
          1         0         1         1         1         1         1
Weighted Avg.  1         0         1         1         1         1
=== Confusion Matrix ===
      a  b  <-- classified as
22  0  |  a = 0
 0 29  |  b = 1

train:/Users/colin/Documents/Study/SE/data_09
train: 461
test: 51
=== Detailed Accuracy By Class ===
          TP Rate  FP Rate  Precision  Recall  F-Measure  ROC Area  Class
          1         0         1         1         1         1         0
          1         0         1         1         1         1         1
Weighted Avg.  1         0         1         1         1         1
=== Confusion Matrix ===
      a  b  <-- classified as
22  0  |  a = 0
 0 29  |  b = 1

```

From the result, we can see that the accuracy is 100%. The program generates a model file at the same time. We can copy this file to BCI Monitor folder, so that the BCI Monitor can load this model easily.



trained.model

##### Extra Deeds in Details Start #####

## Conclusion

In this project, I implemented flexible components to connect an open source EEG signal monitor into the TDR system, so that the TDR can manipulate the data from the sensor for future operation or prediction. The OpenBCI Monitor can give a percentage prediction of meditation status of user

in real time. I also implemented a flexible machine learning module for OpenBCI Monitor. Due to this is the first step of this research, user can modify parameters or select classifiers easily by only modify few parameters...

## Extra Deeds

#### Flexible components in OpenBCI Monitor, user can modify parameters easily to change the behave of the system.

#### Clean data and Feature Extraction from original plain data.

#### Feature selection to select channels for training model as few as possible.

#### Trained multiple classification model for experiments.

#### Provide a flexible machine learning module so that user can change classifier easily for future prediction.

## Appendix

### ApplyModel.java

```
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileReader;
import java.io.ObjectInputStream;
import java.util.ArrayList;

import weka.classifiers.Classifier;
import weka.core.Instance;
import weka.core.Instances;

public class ApplyModel {
    Classifier c = null;
    Instances data = null;
    public ApplyModel(String dataName, String modelFile)
    {
        try {
            data = new Instances(new
BufferedReader(new FileReader(dataName + ".arff")));
            data.setClassIndex(data.numAttributes() - 1);
            c = loadModel(modelFile);
        } catch (Exception e) {
            // TODO Auto-generated catch
block
            e.printStackTrace();
        }

        public int numAttributes() {
            return data.numAttributes();
        }

        public Instance createInstance(ArrayList<double[]>
alData) {
            int step = alData.size();
            int channel = alData.get(0).length;
            Instance instance = new
Instance(numAttributes());
            double[][] data = new double[channel][step];

            for (int i = 0; i < step; i++) {
                double[] datas = alData.get(i);
                for (int j = 0; j < channel; j++) {
                    instance.setValue(i*channel+j, datas[j]);
                    data[j][i] = datas[j];
                }
            }
        }
    }
}
```

```
        }
    }

    for(int m = 0; m < channel; m++) {
        Statistics s = new
Statistics(data[m]);

        instance.setValue(step*channel+m*3+0, s.getMean());

        instance.setValue(step*channel+m*3+1,
s.getVariance());

        instance.setValue(step*channel+m*3+2, s.getStdDev());
    }
    instance.setValue(numAttributes()-1,0);
    return instance;
}

    public double apply(Instance instance) throws Exception
    {
        double[] result =
c.distributionForInstance(instance);
        return result[1];
    }

    private Classifier loadModel(String modelFile) throws
Exception {
        Classifier classifier;

        FileInputStream fis = new
FileInputStream(modelFile);
        ObjectInputStream ois = new ObjectInputStream(fis);

        classifier = (Classifier) ois.readObject();
        ois.close();

        return classifier;
    }
}
```

### CreateFeatureFile.java

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
```

```

import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;

public class CreateFeatureFile {
    String[] files = {

        //"/Users/colin/Documents/Study/SE/test.txt",

        "/Users/colin/Documents/Study/SE/NEG.txt",

        "/Users/colin/Documents/Study/SE/POS.txt",
    };

    public CreateFeatureFile(String dataFile) {
        ArrayList<ArrayList<String>>> features =
new ArrayList<ArrayList<String>>>();
        int step = 4;
        int channel = 16;
        for (int i = 0; i < files.length; i++) {
            // This will reference one line at a
time
            String fileName = files[i];
            String line = null;
            int count = 0;
            double[][] data = new
double[channel][step];
            ArrayList<String> feature = new
ArrayList<String>();
            try {
                // FileReader reads
text files in the default encoding.
                FileReader fileReader
= new FileReader(fileName);

                // Always wrap
FileReader in BufferedReader.
                BufferedReader
bufferedReader = new BufferedReader(fileReader);

                while ((line =
bufferedReader.readLine()) != null) {

                    if(!line.startsWith("%")&&!line.equals("")){

                        //System.out.println(line);

                        String[] datas = line.split(" ");

                        for (int j = 1; j < channel+1; j++) {

                            feature.add(datas[j]);

                            data[j-1][count] = Double.parseDouble(datas[j]);

                        }

                        count++;

                        if (count == step) {

                            for(int m = 0; m < channel; m++) {

                                Statistics s = new Statistics(data[m]);

                                feature.add(s.getMean()+"");

                                feature.add(s.getVariance()+"");

```

```

                                feature.add(s.getStdDev()+"");

                            }

                            feature.add(i+"");

                            features.add(feature);

                        }

                    }

                }

                count = 0;

                data = new double[channel][step];

                feature = new ArrayList<String>();

            }

        }

        // Always close files.

        bufferedReader.close();
    } catch (FileNotFoundException
ex) {

        System.out.println("Unable to open file " + fileName +
        "");

    } catch (IOException ex) {

        System.out.println("Error reading file " + fileName +
        "");

        // Or we could just do
this:

        // ex.printStackTrace();

        System.out.println(count);

    }

    String dataName = dataFile + ".arff";
    try {
        BufferedWriter dataOut = new
BufferedWriter(new FileWriter(dataName));

        dataOut.write("@relation BCI");
        dataOut.newLine();
        dataOut.newLine();

        for (int i = 0; i < step; i++) {
            for (int j = 0; j <
channel; j++) {

                dataOut.write("@attribute Channel_" + j + "_" + i + "
NUMERIC");

                dataOut.newLine();

            }

        }

        for (int i = 0; i < channel; i++) {
            for (int j = 0; j < 3; j++) {

                switch (j) {
                    case 0:

                        dataOut.write("@attribute Mean_" + i + "_" + j + "
NUMERIC");

```

```

        break;

        case 1:

            dataOut.write("@attribute Variance_" + i + "_" + j + "
NUMERIC");

            break;

            case 2:

                dataOut.write("@attribute StdDev_" + i + "_" + j + "
NUMERIC");

                break;

            }

            dataOut.newLine();

        }

        dataOut.write("@attribute

@class@ { '0', '1' }");

        //dataOut.write("@attribute

@class@ NUMERIC");

        dataOut.newLine();
        dataOut.newLine();

        dataOut.write("@data");
        dataOut.newLine();

        for (int i = 0; i < features.size());

        String line = "";
        ArrayList<String> f =

        for (int j = 0; j <

            line = line

        }

        line = line + "" +

        dataOut.write(line);

    }

    dataOut.close();
} catch (IOException e) {
    // TODO Auto-generated catch
    e.printStackTrace();
}

block
}

}

```

## Statistics.java

```

import java.util.Arrays;

public class Statistics {
    double[] data;
    int size;

    public Statistics(double[] data) {
        this.data = data;
        size = data.length;
    }

    double getMean() {
        double sum = 0.0;
        for (double a : data)
            sum += a;
        return sum / size;
    }

    double getVariance() {
        double mean = getMean();
        double temp = 0;
        for (double a : data)
            temp += (a - mean) * (a - mean);
        return temp / size;
    }

    double getStdDev() {
        return Math.sqrt(getVariance());
    }

    public double median() {
        Arrays.sort(data);

        if (data.length % 2 == 0) {
            return (data[(data.length / 2) - 1]
+ data[data.length / 2]) / 2.0;
        }
        return data[data.length / 2];
    }
}

```

## TrainModel.java

```

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.ObjectOutputStream;

import weka.classifiers.Classifier;
import weka.classifiers.functions.SMO;
import weka.core.Instances;

public class TrainModel {

    public TrainModel(String dataFile, String modelFile)
throws Exception {

        String dataName = dataFile;
        Instances data = new Instances(new
BufferedReader(new FileReader(dataName + ".arff")));
        data.setClassIndex(data.numAttributes() - 1);

        SMO c = new SMO();
        c.buildClassifier(data);

        saveClassifier(c, modelFile);
    }
}

```

```

    }

    public void saveClassifier(Classifier c, String modelFile)
        throws IOException {

        // serialize model
        ObjectOutputStream oos = new
ObjectOutputStream(
        new
FileOutputStream(modelFile));
        oos.writeObject(c);
        oos.flush();
        oos.close();
    }
}

```

## TrainTestModel.java

```

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.Random;

import weka.core.Instances;

public class TrainTestModel {

    public TrainTestModel(String dataFile) throws
Exception {
        int seed = 0; // the seed for randomizing the
data
        int folds = 10; // the number of folds to
generate, >=2
        int runs = 1;
        String dataName = dataFile;
        Instances data = new Instances(new
BufferedReader(new FileReader(dataName + ".arff")));
        data.setClassIndex(data.numAttributes() - 1);

        Instances randData = null;
        for (int i = 0; i < runs; i++) {
            seed = i + 1;
            Random rand = new
Random(seed); // create seeded number generator

            randData = new Instances(data);
// create copy of original data

            randData.randomize(rand); //
randomize data with number generator
            //randData.stratify(folds);

            for (int n = 0; n < folds; n++) {
                String train =
Instances trainset =
randData.trainCV(folds, n);
                Instances testset =
randData.testCV(folds, n);

                System.out.println("train:" + train);

```

```

                System.out.println("train: " + trainset.numInstances());
                System.out.println("test: " + testset.numInstances());

                WekaWrapper
wekaWrapper = new WekaWrapper();
                wekaWrapper.labelFile
= train + ".label";

                wekaWrapper.TrianTest(trainset, testset);
            }
        }
    }
}

```

## CreateBCIUploader.java

```

System.out.println("+++++++");
System.out.println(purpose);

System.out.println("+++++++");
String sUId = kvList.getValue("uid");
String[] sChannels = new
String[Integer.parseInt(kvList.getValue("channels"))];
for (int i = 0; i < sChannels.length; i++) {
    sChannels[i] =
kvList.getValue("channels"+i);
}

String sDate = kvList.getValue("datetime");
String sOriginator = kvList.getValue("originator");

if (sUId != null && !sUId.equals("")) {
    // TODO for testing, I don't need to change the uid
here.
    reading.uid = sUId;
}
reading.channels = new
double[Integer.parseInt(kvList.getValue("channels"))];
for (int i = 0; i < reading.channels.length; i++) {
    if (sChannels[i] != null
&& !sChannels[i].equals("")) {
        reading.channels[i] =
Double.parseDouble(sChannels[i]);
    }
}

if (sDate != null && !sDate.equals(""))
{
    reading.dateBCISensor = Long.parseLong(sDate);
}
if (sOriginator != null && !sOriginator.equals("")){
    reading.originator = sOriginator;
}

System.out.println(sUId);
for (int i = 0; i < sChannels.length; i++) {
    System.out.println(sChannels[i]);
}
System.out.println(sDate);
System.out.println(sOriginator);

```

```

    }
    case "BCIMonitor":

//execute(formQuery(reading.dateVotes,"SocialNetwork","votes",reading.votes));
        if (reading.purpose.equals("upload")) {
            for (int i = 0; i < reading.channels.length; i++) {

                execute(formQuery(reading.dateBCISensor,reading.up,
"channel"+i,reading.channels[i], reading.originator));
            }

        }

        break;

CreateBCIMonitor.java

try {
    am = new ApplyModel(data,
model);
} catch (Exception e) {
    // TODO Auto-generated catch
block
    e.printStackTrace();
}

private static void componentTask() {
    try {

        // TODO: create your component task here
        uploadBCI();
        time =
System.currentTimeMillis();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

long currentTime = System.currentTimeMillis();
long diffTime =
currentTime - time;

//System.out.println(diffTime);

String[] sChannels =
new String[Integer.parseInt(kvList.getValue("channels"))];
for (int i = 0; i < sChannels.length; i++) {
    sChannels[i] =
kvList.getValue("channels"+i);
}

reading.channels = new
double[Integer.parseInt(kvList.getValue("channels"))];
for (int i = 0; i < reading.channels.length; i++) {
    if (sChannels[i] != null
&& !sChannels[i].equals("")) {
        reading.channels[i] =
Double.parseDouble(sChannels[i]);
    }
}

if (alData.size() < 4) {

    alData.add(reading.channels);
}

if (alData.size() ==
4){

    analysis();
    alData =
new ArrayList<double[]>();
}

```

```

        if (diffTime > 15000)

{

    componentTask();
}

break;
private static void analysis () {
    try{
        Instance instance =
am.createInstance(alData);

        System.out.println(am.apply(instance));
    } catch (Exception e) {
        System.out.println("Error in analysis");
    }
}
}

```

## BCIReading.pde

```

class BCIReading {
    double[] channels;
    String up = "";
    String purpose = "";
    String originator = "";
    long date;

    String uid = "376896";
    public BCIReading() {
    }
}

```

## CreateBCISensor.pde

```

collect(data.values, scale_to_uV, nchan);
    reading.date = System.currentTimeMillis();
    //reading
    record.putPair("uid", reading.uid + "");
    record.putPair("channels", reading.channels.length+"");
    for (int i = 0; i < reading.channels.length; i++) {
        record.putPair("channels"+i, reading.channels[i]+"");
    }
    record.putPair("datetime", reading.date + "");
    //send reading message to GUI, uploader
    //encoder.sendMsg(record);
    //send reading message to controller
    // record.putPair("Receiver", "BCIController");
    // send to BCIMonitor
    record.putPair("Receiver", "BCIMonitor");
    record.putPair("originator", "machine");
    record.putPair("Purpose", "Reading");
    encoder.sendMsg(record);
    record.removePair("Receiver");
    encoder.sendMsg(record);
void collect(int[] values, float scale_fac, int nchan) {
    reading.channels = new double[nchan];
    int nVal = values.length;
    for (int lval = 0; lval < nVal; lval++) {
        reading.channels[lval] = scale_fac * (float)(values[lval]);
        //System.out.println(reading.channels[0]);
    }
}
}

```